Express Mail EL480513625US
Date of Deposit: Nov. 22, 2000

METHODOLOGY FOR IMPROVING TCP THROUGHPUT OVER LOSSY COMMUNICATION LINKS

Filed of the invention:

This invention relates to a methodology for improving TCP throughput over lossy communication links without affecting performance over non-lossy links.

Background of the invention:

Transmission Control Protocol (TCP) [1,4,10], the transport layer protocol widely used in the internet has gradually evolved over the years with newer mechanisms that have helped improve its performance. TCP has been designed as a stream based reliable transfer protocol that can work on top of an underlying network protocol. The built-in end-to-end congestion control and reliable data transfer mechanisms minimize the complexity within the network and more significantly are a critical factor in the robustness of the internet.

Throughout the evolution of TCP, a loss in the network has always been an indication of congestion. This is not the case in many situations, especially when the connection spans lossy links. Due to this, TCP performance is good over wire-line internet links but degrades over lossy links [5,7], such as wireless and satellite channels. Losses due to link characteristics cause TCP to reduce its sending rate thus resulting in bandwidth under-utilization. Wireless links for example, are often characterized by high bit error rates due to channel fading, noise, interference, and intermittent connectivity due to hand-offs. TCP performance in such networks suffers from throughput degradation and very high interactive delays, because the sender misinterprets random packet losses as congestion.

Various mechanisms have been put forward to alleviate the affects of non congestive losses on TCP performance over lossy links. These mechanisms adopt a variety of schemes such as Forward Error Correction, split TCP connections, local retransmissions, link aware protocols, Explicit Loss Notification (ELN). A brief description of some mechanisms is provided.

(1) **Split connection protocol**: [6] This approach splits the TCP connection between the sender and the receiver into two separate connections, one between the sender and the base station and the other between the same base and the receiver. A base station is a router sitting at the boundary of a lossy network. A specialized protocol fine tuned to the link characteristics is used over the lossy hops to reduce the problems faced by standard TCP over lossy links. The major drawback of the system though, is that it violates end-to-end semantics. Acknowledgements for packets may reach the sender before the data actually reaches the receiver. This also makes the protocol vulnerable to base station crashes.

(2) **Snoop Protocol**: [8] Snoop protocol is basically a link aware protocol. The mechanism uses an agent sitting on the base station that monitors TCP packets in both directions and caches them. It retransmits such packets if it detects their loss either through duplicate acknowledgements or a local timeout for the acknowledgement.

(3) **Explicit Loss Notification**: [12] ELN is a link aware mechanism using which the cause for the loss of a packet can be communicated to the TCP sender. The ‘snoop’ agent sitting at a ‘base’ station monitors the traffic and keeps track of holes in the sequence space of every connection as it receives data segments. A hole is a missing interval in the sequence space. It then sets the

ELN bit on acknowledgements, especially duplicate acknowledgements, if they correspond to a segment in the list. The source uses the ELN notification in its congestion detection mechanism. All such snoop protocols have the inherent disadvantage of being incompatible with end-to-end flows that employ IP layer security using encryption or other means.

(4) **Patent US5974028**: The methodology for providing transport protocol within a communication network having a lossy link. The receiver distinguishes between packets received with non-congestion bit errors and packets not received due to congestion. It sends selective acknowledgement to the sender indicating which packets were received with non-congestion bit-errors, while suppressing the duplicate acknowledgements to avoid invocation of congestion control mechanisms.

The object and summary of the invention:

The object of this invention is to obviate the above disadvantages and provide a system for improving the TCP throughput over lossy communication links without affecting performance over non-lossy links.

To achieve the said objective this invention relates to a system for improving TCP throughput over lossy communication links without affecting performance over non-lossy links comprising:

- means for determining lookahead-loss which is the number of lost packets in a given **loss-window.**
- means for using **said loss-window** and said lookahead loss to detect congestion in said communication links, and
- means for controlling transmission under congestion conditions as well

as under normal conditions.

The said means for determining lookahead-loss is a mechanism for identifying the number of packets transmitted by the sender in said **loss-window**, for which either of the following conditions is true:

- sender has received at least max-**dupacks** (an appropriately selected number, typically three) duplicate cumulative acknowledgements,
- sender has neither received **acknowledgement** nor selective **acknowledgement** for said packets, while it has received selective acknowledgements for at least max-dupsacks (an appropriately selected number, typically three) packets with higher sequence numbers.

The said means for detecting congestion is a mechanism for identifying when the number of packets lost in a loss-window is greater than an appropriately selected preset number.

The said means for controlling transmission is a TCP k-SACK protocol which is a modification of the fast retransmit algorithm of the basic congestion control algorithm of TCP to include:

- entering a 'halt growth phase' whenever lookahead loss is greater than zero and congestion is not detected,
- entering a 'k-recovery phase' whenever the congestion is detected.

In the said 'halt growth phase', the sender freezes the congestion window and maintains it in that state.

Entry into the said **'k-recovery phase'** reduces the congestion window to half its original size, while the slow-start threshold is reduced to half only on the first occasion of entry into the k-recovery phase during a packet loss recovery cycle.

The above system further includes:

- "Post Recovery" wherein the sender continues in congestion avoidance or slow-start phase at the end of the fast recovery phase,
- more accurate estimation of pipe size using the received selective acknowledgement (SACK) data,
- use of said accurate pipe size information for controlling window inflation and deflation thereby allowing **quicker retransmission** of lost packets and resulting faster recovery.

The present invention further provides a method for improving TCP throughput over lossy communication links without affecting performance over non-lossy links comprising:

- determining lookahead-loss which is the number of lost packets in a given **loss-window.**
- using said **loss-window** and said lookahead loss to detect congestion in said communication links, and
- controlling transmission under congestion conditions as well as under normal conditions.

The said determining of lookahead-loss is for identifying the number of packets transmitted by the sender in said **loss-window**, for which either of the following conditions are true:

- sender has received at least max-**dupacks** (an appropriately selected number, typically three) duplicate cumulative acknowledgements,
- sender has neither received **acknowledgement** nor selective **acknowledgemnt** for said packets, while it has received selective acknowledgements for at least max-dupsacks (an appropriately selected number, typically three) packets with higher sequence numbers.

The said detecting of congestion is for identifying when the number of packets lost in a loss-window is greater than or equal to an appropriately selected preset number.

The said controlling of transmission is a TCP k-SACK protocol which is a modification of the fast retransmit algorithm of the basic congestion control algorithm of TCP to include:

- entering a 'halt growth phase' whenever lookahead loss is greater than zero and congestion is not detected,
- entering a 'k-recovery phase' whenever the congestion is detected.

During said 'halt growth phase', the sender freezes the congestion window and maintains it in that state.

During said 'k-recovery phase' reduces the congestion window to half its original size, while the slow-start threshold is reduced to half only on the first occasion of entry into the k-recovery phase during a packet loss recovery cycle.

The above method further includes:

- "Post Recovery" wherein the sender continues in congestion avoidance or slow start phase at the end of the fast recovery phase,
- more accurate estimation of pipe size using the received selective acknowledgement (SACK) data,
- use of said accurate pipe size information for controlling window inflation and deflation thereby allowing early retransmit of lost packets and resulting faster recovery.

The instant invention also provides a computer program product comprising computer readable program code stored on computer readable storage medium embodied therein for improving TCP throughput over lossy communication links without affecting performance over non-lossy links comprising:

- computer readable program code means configured for determining lookahead-loss which is the number of lost packets in a given **loss-window**.
- computer readable program code means configured for using said **loss-window** and said lookahead loss to detect congestion in said communication links, and
- computer readable program code means configured for controlling transmission under congestion conditions as well as under normal conditions.

The said computer readable program code means configured for determining lookahead-loss is a mechanism for identifying the number of packets transmitted by the sender in **said loss-window**, for which either of the following conditions is true:

- sender has received at least **max-dupacks** (an appropriately selected number, typically three) duplicate cumulative acknowledgements,
- sender has neither received **acknowledgement** nor selective **acknowledgement** for said packets, while it has received selective acknowledgements for at least max-dupsacks (an appropriately selected number, typically three) packets with higher sequence numbers.

The said computer readable program code means configured for detecting congestion is a mechanism for identifying when the number of packets lost in a loss-window is greater than an appropriately selected preset number.

The said computer readable program code means configured for controlling transmission is a TCP k-SACK protocol which is a modification of the fast retransmit algorithm of the basic congestion control algorithm of TCP to include:

- entering a 'halt growth phase' whenever lookahead loss is greater than zero and congestion is not detected,
- entering a 'k-recovery phase' whenever the congestion is detected.

During said 'halt growth phase', the sender freezes the congestion window and maintains it in that state.

During said 'k-recovery phase' reduces the congestion window to half its original size, while the slow-start threshold is reduced to half only on the first occasion of entry into the k-recovery phase during a packet loss recovery cycle.

The above computer program product further includes:

- "Post Recovery" wherein the sender continues in congestion avoidance or slow start phase at the end of the fast recovery phase,
- more accurate estimation of pipe size using the received selective acknowledgement (SACK) data,
- use of said accurate pipe size information for controlling window inflation and deflation thereby allowing early retransmit of lost packets and resulting faster recovery.

Brief Description of the Drawings:

The invention will now be described with reference to accompanying drawings.

Fig 1 shows a Sample cwnd evolution for k-SACK.

Fig. 2 shows the flow diagram of congestion control algorithm

Fig. 3 shows k-SACK fast recovery cwnd and ssthresh dynamics.

Fig. 4 shows the network topology gateways, G1 & G2 sitting on opposite sides of the bottleneck link with a bandwidth of 10 Mbps and 5 ms delay.

Fig. 5a shows the completion time for 100 kb transfer on Drop Tail gateway.

Fig. 5b shows the completion for 100 kb transfer on RED gateway.

Fig. 6a shows the completion time for 1 Mb transfer on Drop Tail gateway.

Fig. 6b shows the completion time for 1 Mb transfer on RED gateway.

Fig. 7a shows the completion time for 10 Mb transfer on Drop Tail gateway.

Fig. 7b sows the completion time for 10 Mb transfer on RED gateway.

Fig. 8 shows throughput performance across non lossy link

Fig. 9 shows throughput performance across lossy link (1% packet loss rate)

Fig. 10 shows throughput performance across lossy link (3% packet loss rate)

Fig. 11 shows throughput performance across lossy link (5% packet loss rate)

Detailed Description of the Drawings:

Figure 1 shows a sample lwnd evolution for k-SACK for a case with a k-value of 2'. The system window increases rapidly in the slow start phase (1) until it reaches 'ssthresh'. At this stage the system switches to 'congestion avoidance' phase and the window increases slowly (2). When a single packet loss event occurs 'event 1', the system enters 'halt growth' phase (3). In this state the sender continues in fast recovery but does not change the loss-window 'lwnd' and slow start threshold 'ssthresh' values. This situation continues upto 'event 3' i.e. single loss recovered. At this point all the lost packets have been recovered and the system reverts to 'congestion avoidance' phase (4). When another packet loss is detected, the system once again enters the 'halt growth' phase (5) and maintains until 'event 2' i.e. two losses detected. At this point the packet losses equal the 'k' factor and therefore the system goes into the 'k-recovery' phase (6). In this phase the loss-window 'lwnd' ad 'ssthresh' are immediately decreased to half the current 'lwnd' value. The loss-window growth continues from this point and the system remains in 'k-recovery' phase until 'event 1' once again occurs at which point the system enters halt growth phase (7). The system remains in this phase until 'event 3' at which point the packet loses have once again been recovered and the system enters 'congestion avoidance' phase (8). At (9) another 'event 1' occurs and the system reverts to 'halt growth'. This state is maintained until 'event 3' occurs and the system reverts to 'congestion avoidance' phase (10). Another 'event 1' occurs at (11) and takes the system into 'halt growth' phase which is maintained until (12) when an 'event 3' results in the recovery of loss packets and brings the system back to 'congestion avoidance' phase. At (13) an 'event 2' (loss of two packets) occurs and causes the system to go into 'k-recovery' phase. The loss window once again collapses to half the current value and the 'ssthresh' is adjusted to become equal to new 'lwnd'. Another 'event 3' occurs at (14) and brings to system onto 'halt growth' phase until the occurrence of another event 3 at (15) results in full recovery and reverts it to 'congestion avoidance' phase. Finally, an 'event 1' at (16) reverts the system to 'halt growth' phase.

The basic congestion control algorithms of TCP [1,13], *slow start, congestion avoidance, fast retransmit* are used in the protocol. k-SACK uses a modified fast recovery algorithm.

k-SACK Fast Recovery: The fast recovery phase is separated into two different phases for the purpose of comprehension. On entering the fast recovery phase, if the *lookahead-loss* is less than k, the sender enters the 'halt growth phase' of fast recovery, else it moves to the 'k-recovery phase' of fast recovery.

- halt growth Phase: When in this state, the sender continues in fast recovery, but does not change the lwnd and ssthresh values. It freezes the congestion window and maintains it in that state until the *lookahead-loss* becomes at least k, whereupon the sender enters the 'k-recovery Phase'. Congestion window growth is unfrozen on moving to 'k-recovery Phase'. After receiving an acknowledgement for up to or beyond 'recover', this phase is exited and the growth of the congestion window unfrozen.
- k-recovery Phase: A TCP sender enters this phase while in fast recovery, if the *lookahead-loss* is at least k. Upon entering this phase for the first time while in the same fast recovery, the ssthresh is set to max(cwnd/2, 2*MSS),and the congestion window is set to ssthresh. When *lookahead-loss*

becomes less than k, the sender changes state to the 'halt growth phase'. The sender may re-enter the k-recovery phase after making a transition to the halt growth phase, while within the same fast recovery. On such a re-entry, cwnd is set to max(cwnd/2, 1MSS) and ssthresh is not changed. The sender continues in this phase until either the *lookahead-loss* is at least k or the completion of fast recovery.

Figure 2 shows the system operation. When no losses are detected, the source is in slow start (SS) / congestion avoidance (CA) phase (2.1). Whenever an acknowledgement packet is received (2.2), the source checks for a timeout condition (2.3). If a timeout is detected the system reverts to SS/CA phase. If however, no timeout is detected the system calculates the lookahead-loss (2.4) and then checks for packet loss (2.5). If no packet loss is detected the source remains in SS/CA phase. If a packet loss is detected the source enters the 'k-sack' fast recovery phase (2.6) and computes the lookahead loss (2.7).

Whenever the lookahead loss is found to be greater than '0' but less than 'k' the system moves to the 'halt growth' phase (2.8). Once in the 'halt-growth' phase the source checks for acknowledgement (2.9). If no acknowledgement is received the system checks for a timeout condition (2.10). If a timeout is detected the system reverts to SS/CA phase. If no timeout is detected the system reverts to check for acknowledgement (2.9). Once an acknowledgement is received the system the system computes the lookahead loss (2.11). If the lookahead loss is '0' the fast recovery is complete (2.13) and the source reenters the SS/CA phase (2.1) else it once again compares the lookahead loss with 'k' (2.7).

When the lookahead loss is found to be greater than 'k' the system enters the 'k-recovery' phase (2.13). Once in 'k-recovery' the system checks for acknowledgement (2.14). If no acknowledgement is received the system checks for a timeout condition (2.15). If a timeout is detected the system reverts to SS/CA phase (2.1). If no timeout is detected the system reverts to check for acknowledgement (2.14). Once an acknowledgement is received the system the system computes the lookahead loss (2.16). If the lookahead loss is '0' the fast recovery is complete (2.17) the source reenters the SS/CA phase (2.1) else it once again compares the lookahead loss with 'k' (2.7).

In figure 3, the system comprises three states:

The "slowstart/congestion avoidance state (0)", the 'Halt Growth State (H)" and the "k-recovery state (K)".

The system initially starts in the '(O)' state and continues in tat state until the occurrence of a packet loss. If the number of lost packets is < 'k', the system transitions to the 'H' state and while doing so freezes the loss window 'lwnd'. If on the other hand, the number of lost packets is ≥ 'k' the system transitions to the 'K' state and while doing so sets the loss window 'lwnd' to 'lwnd/2' and the slow start threshold 'ssthresh' t the same values.

Once in the 'H' state the system remains in that state until either all the lost packets have been recovered at which point the system reverts to the 'O' state, or the number of lost packets increase to ≥ 'k' at which point the system transitions to the 'K' state. When reverting to the 'O' state the system unfreezes the 'lwnd'. If however, the system transitions to the 'k' state it updates the 'lwnd' to the value 'lwnd/2', while the 'lwnd' growth remains unfrozen. The 'ssthresh' value is set to 'lwnd/2' on the first transition to the 'K' state while it is unchanged in subsequent transitions within a cycle of recovery from packet loss.

Once in the 'K' state the system remains in tat sate until packet loss > = 'k', the system transition to the 'O' state and while doing so $k_{recover} = 0$, no loss is detected.

If on other hand, the number of recovered packet losses > k the system transition to the 'H' state and while doing so it freezes the loss window 'lwnd'.

Figure 4 shows the network topology gateways, G1 & G2 sitting on opposite sides of the bottleneck link with a bandwidth of 10 Mbps and 5 ms delay. The other links are very high capacity links. S1 through S6 are TCP sources and R1 through R6 are the corresponding sinks. To simulate the affect of non-TCP traffic in the internet, the nodes U1 and U2 are shown as a source and sink, the source sending Paretodistributed UDP traffic to the sink through the bottleneck link. The bottleneck link is modeled as a lossy link with varying packet loss rates. The default packet size of 576 bytes is used to carry the data.

Figure 5a, 5b, 6a, 6b, 7a, 7b shows the simulation results for a fixed file transfers of size 100 kb (short transfer), 1Mb (medium transfer) and 10 Mb (large transfer) across a bottleneck link

For these simulations, the bottleneck link also carries 15-20 % Pareto distributed UDP traffic. The experiments have been conducted using droptail as well as RED

[2] gateways at the boundaries of the bottleneck link. Traffic flows in one direction only and hence acknowledgements are never dropped due to congestion on the reverse path. The time taken is averaged over six simultaneous connections and also over multiple simulation runs. Other than due to random losses, the simulation behavior as been forced to change over multiple runs by varying the starting times of the TCP connections.

The completion times for different file sizes clearly show the superior performance of K-SACK over TCP SACK. For non-lossy links, the performance of the different variants are almost similar. However for close to 5% packet loss rates, the completion times are almost half as compared to those observed for TCP SACK. The increase in performance is observed across both droptail as well as RED gateways.

Figs. 8, 9, 10 & 11 show the throughput performance across non-lossy link. Figures show the throughput of each variant for a very long (persistent source) data transfer. In this case, we have used six simultaneous connections all of the same variant at a time. The throughput values averaged over multiple runs are representative of the link utilization by the TCP senders. The throughput plot of a random TCP connection is plotted for different error rates. As can be seen in the graphs, the throughput of K-SACK is considerably higher than that for TCP SACK even for small error rate and the difference in their respective throughputs increases significantly as the random error rate increases, since TCP KSACK maintains reasonable performance when random losses occur while TCP SACK's performance degrades drastically.

TERMINOLOGY:

(1) **cwnd**: The congestion window of a TCP sender.

(2) **slow start**: The initial phase that a TCP sender starts transmission. cwnd is incremented by one for each acknowledgement received in this phase.

(3) **ssthresh**: The slow start threshold of a TCP sender.

(4) **congestion avoidance**: The phase a TCP sender enters when the cwnd becomes greater than ssthresh. The cwnd is incremented by one for every successful transmission of cwnd number of packets.

(5) **MSS**: The maximum segment size.

(6) **max-dupacks**: The number of duplicate acknowledgements that the TCP sender must receive for a packet to conclude that it is lost.

(7) **fast recovery**: The Newreno mechanism to recover from a packet loss.

(8) **pipe**: The measure of the number of outstanding packets that a TCP sender has in the link.

(9) **ns-2**: The network simulator version-2 [14].

(10) **Lookahead-loss**: *Lookahead-loss* is defined to be equal to the number of packets transmitted by the sender in the given loss-window, such that for all such packets, either of the two holds:

- Sender has received at least max-dupacks (typically three) duplicate cumulative acknowledgements for the packet
- The sender has received selective acknowledgements for at least max-dupsacks (typically three) packets with higher sequence numbers.

11. **Max-dupsacks**: the maximum number of selective acknowledgement received for packets with higher sequence numbers for it to contribute to the look-ahead loss

12. **lwnd**: loss-window: the number of transmitted packets used for computing the look-ahead loss.

REFERENCES:

[1] W.Richard Stevens TCP/IP Illustrated, Volume 1: The Protocols. Addison Wesley, 1994.

[2] Sally Floyd and Van Jacobson. ``Random Early Detection Gateways for Congestion Avoidance,''.IEEE/ACM Transactions on Networking, 1(4), pp. 397--413, Jan. 1998.

[3] M.Mathis, J.Mahdavi S.Floyd and A.Romanow.``TCP Selective Acknowledgement Options," 1996. RFC-2018.

[4] V.Jacobson.``Congestion Avoidance and Control,".ACM SIGCOMM 1988. pp 314-329, Aug 1988.

[5] K.Fall and S.Floyd.``Comparison of Tahoe, Reno and Sack TCP". March 1996.

[6] A.Bakre and B.R.Badrinath.``I-TCP: Indirect TCP for Mobile Hosts,".Proc. 15th International Conference on Distributed Computing Systems, May 1995.

[7] Anurag Kumar.``Comparative Performance Analysis of Versions of TCP in a Local Network with Lossy Link,".IEEE/ACM Transactions on Networking, Dec.1996.

[8] H.Balkrishnan, V.N.Padmanabhan, S.Seshan, M.Stemm, E.Amir and R.H.Katz,``TCP Improvements for Heterogeneous Networks: The Daedalus Approach". Proc. 35th Annual Allerton Conference on Communication, Control and Computing, Urbana, Illinois. Oct 1997.

[9] E.Ayonglu S.Paul, T.F.LaPorta, K.K.Sabnani and R.D.Gitlin.``AIRMAIL: A Link-Layer Protocol for Wireless Networks,".ACM ACM/Baltzer Wireless Networks Journal,1: pp 47-60, Feb 1995.

[10] J.B.Postel.``Transmission Control Protocol,".RFC, Information Sciences Institute, Marina Del Rey, CA, Sept 1981. RFC-793.

[11] S.Floyd and T.R.Henderson.``The Newreno modification to TCP's Fast Recovery Algorithm,".RFC 2582, April 1999.

[12] H.Balakrishnan and R.H. Katz.``Explicit Loss Notification and Wireless Web Performance,".Proc. IEEE Globecom 1998, Internet mini-conference, Sydney, Australia.

[13] ns-2 simulator, *http://www.isi.edu/nsnam/ns*.